Formale Spezifikation und Verifikation

Mirco Tribastone

Institut für Informatik
Ludwig-Maximilians-Universität München
tribastone@pst.ifi.lmu.de

Process Algebras

Calculus of Communicating Systems

CCS Basics

Sequential Fragment

- **0** process (the only atomic process)
- Action prefixing (*a.P*)
- Names and recursive definitions ($\triangleq$)
- Nondeterministic choice ($+$)

Parallelism and Renaming

- Parallel composition (operator $\mid$) for synchronous communication between two components (*handshake synchronization*)
- Restriction ($P \backslash L$)
- Relabelling ($P[f]$)

Channels, Actions, Process Names)

Let

- $\mathcal{A}$ be a set of channel names (sometimes, simply called names).
  For instance, *tea* and *coffee* are channel names.
- $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$ be a set of labels where
  - $\overline{\mathcal{A}} = \{\overline{a} \mid a \in \mathcal{A}\}$ (elements of $\overline{\mathcal{A}}$ are called co-names)
  - by convention $\overline{\overline{a}} = a$
  - $\tau \notin \mathcal{A}$
- $Act = \mathcal{L} \cup \{\tau\}$ is the set of actions where
  - τ is the internal or silent action

  (e.g. τ, *tea*, $\overline{coffee}$ are actions)
- $\mathcal{K}$ is a set of process names (or constants) (usually with upper-case initials).

Definition of CCS (expressions)

$P :=$	K	$\mid$	Process constants ($K \in \mathcal{K}$, $K \triangleq P$)
	$\alpha.P$	$\mid$	Prefix ($\alpha \in Act$)
	$\sum_{i \in I} P_i$	$\mid$	Summation (I is an arbitrary index set)
	$P_1 \mid P_2$	$\mid$	Parallel composition
	$P \setminus L$	$\mid$	Restriction ($L \subseteq \mathcal{A}$)
	$P[f]$	$\mid$	Relabelling ($f : Act \rightarrow Act$) such that ■ $f(\tau) = \tau$ ■ $f(\overline{a}) = \overline{f(a)}$

The set of all terms generated by the abstract syntax is the set of CCS process expressions (and is denoted by $\mathcal{P}$).

Notation

$P_1 + P_2 = \sum_{i \in \{1,2\}} P_i$ $\qquad$ $Nil = \mathbf{0} = \sum_{i \in \emptyset} P_i$

Precedence

Precedence

1. Restriction and relabelling (tightest binding)
2. Action prefixing
3. Parallel composition
4. Summation

Example

$$R + a.P \mid b.Q\backslash L \quad \text{means} \quad R + \big((a.P) \mid (b.(Q\backslash L))\big) \ .$$

Defining Equations

CCS program

A collection of defining equations of the form

$$K \triangleq P$$

where $K \in \mathcal{K}$ is a process constant and $P \in \mathcal{P}$ is a CCS process expression.

- Only one defining equation per process constant.
- Recursion is allowed: e.g. $A \triangleq \overline{a}.A \mid A$.

Structural Operational Semantics for CCS

Given a collection of CCS defining equations, we define the following LTS $(Proc, Act, \{\xrightarrow{a} \mid a \in Act\})$:

- $Proc = \mathcal{P}$ (the set of all CCS process expressions)
- $Act = \mathcal{L} \cup \{\tau\}$ (the set of all CCS actions including τ)
- The transition relations are given by SOS rules of the form:

$$\text{RULE NAME} \quad \frac{\textit{premises}}{\textit{conclusion}} \quad \textit{side conditions}$$

SOS Rules for CCS ($\alpha \in Act$, $a \in \mathcal{L}$)

ACT $\dfrac{}{\alpha.P \xrightarrow{\alpha} P}$

SUM$_j$ $\dfrac{P_j \xrightarrow{\alpha} P'_j}{\sum_{i \in I} P_i \xrightarrow{\alpha} P'_j} \quad j \in I$

COM1 $\dfrac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q}$

COM2 $\dfrac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'}$

COM3 $\dfrac{P \xrightarrow{a} P' \quad Q \xrightarrow{\bar{a}} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$

RES $\dfrac{P \xrightarrow{\alpha} P'}{P \setminus L \xrightarrow{\alpha} P' \setminus L} \quad \alpha, \overline{\alpha} \notin L$

REL $\dfrac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]}$

CON $\dfrac{P \xrightarrow{\alpha} P'}{K \xrightarrow{\alpha} P'} \quad K \triangleq P$

Deriving Transitions in CCS

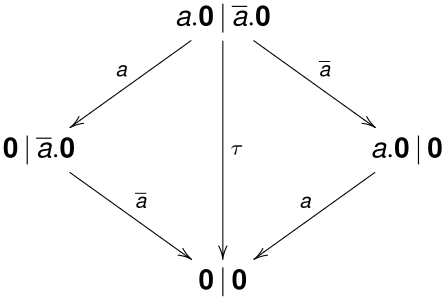
Let $A \triangleq a.A$. Then

$$\big((A \,|\, \overline{a}.\mathbf{0}) \,|\, b.\mathbf{0}\big)[c/a] \xrightarrow{c} \big((A \,|\, \overline{a}.\mathbf{0}) \,|\, b.\mathbf{0}\big)[c/a].$$

Why?

$$
\text{REL}\;\dfrac{\text{COM1}\;\dfrac{\text{COM1}\;\dfrac{\text{CON}\;\dfrac{\text{ACT}\;\dfrac{}{a.A \xrightarrow{a} A}}{A \xrightarrow{a} A}\,A \triangleq a.A}{A \,|\, \overline{a}.\mathbf{0} \xrightarrow{a} A \,|\, \overline{a}.\mathbf{0}}}{(A \,|\, \overline{a}.\mathbf{0}) \,|\, b.\mathbf{0} \xrightarrow{a} (A \,|\, \overline{a}.\mathbf{0}) \,|\, b.\mathbf{0}}}{\big((A \,|\, \overline{a}.\mathbf{0}) \,|\, b.\mathbf{0}\big)[c/a] \xrightarrow{c} \big((A \,|\, \overline{a}.\mathbf{0}) \,|\, b.\mathbf{0}\big)[c/a]}
$$

LTS of the Process $a.\mathbf{0} \mid \overline{a}.\mathbf{0}$



$$
\begin{aligned}
Proc &= \{a.\mathbf{0} \mid \overline{a}.\mathbf{0}, \mathbf{0} \mid \overline{a}.\mathbf{0}, a.\mathbf{0} \mid \mathbf{0}, \mathbf{0} \mid \mathbf{0}\} \ , \\
Act &= \{a, \overline{a}, \tau\} \ , \\
\xrightarrow{a} &= \big\{(a.\mathbf{0} \mid \overline{a}.\mathbf{0}, \mathbf{0} \mid \overline{a}.\mathbf{0}), (a.\mathbf{0} \mid \mathbf{0}, \mathbf{0} \mid \mathbf{0})\big\}, \\
\xrightarrow{\overline{a}} &= \big\{(a.\mathbf{0} \mid \overline{a}.\mathbf{0}, a.\mathbf{0} \mid \mathbf{0}), (\mathbf{0} \mid \overline{a}.\mathbf{0}, \mathbf{0} \mid \mathbf{0})\big\} \ , \\
\xrightarrow{\tau} &= \big\{(a.\mathbf{0} \mid \overline{a}.\mathbf{0}, \mathbf{0} \mid \mathbf{0})\big\} \ .
\end{aligned}
$$

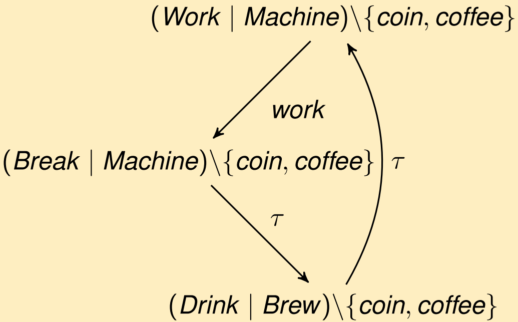
Using Restriction

LTS of $(a.\mathbf{0} \mid \overline{a}.\mathbf{0})\backslash\{a\}$

$$(a.\mathbf{0} \mid \overline{a}.\mathbf{0})\backslash\{a\} \xrightarrow{\tau} (\mathbf{0} \mid \mathbf{0})\backslash\{a\}$$

Another Example

$$
\begin{aligned}
Work &\triangleq work.Break \\
Break &\triangleq \overline{coin}.Drink \\
Drink &\triangleq coffee.Work \\
Machine &\triangleq coin.Brew \\
Brew &\triangleq \overline{coffee}.Machine
\end{aligned}
$$



Equivalence Relations

Definition

Let S be a set. A binary relation $R \subseteq S \times S$ is called an equivalence relation if the following hold:

- R is reflexive, i.e., it holds that $\langle s, s \rangle \in R$ for all $s \in S$
- R is symmetric, i.e., if $\langle s_1, s_2 \rangle \in R$ then $\langle s_2, s_1 \rangle \in R$ for all $s_1, s_2 \in S$
- R is transitive, i.e., if $\langle s_1, s_2 \rangle \in R$ and $\langle s_2, s_3 \rangle \in R$ then $\langle s_1, s_3 \rangle \in R$ for all $s_1, s_2, s_3 \in S$

A binary relation that is reflexive and transitive is called a preorder.

Convention

It is customary to write $s_1 \, R \, s_2$ to indicate $\langle s_1, s_2 \rangle \in R$

Behavioural Equivalence

Implementation

$$CM \triangleq coin.\overline{coffee}.CM$$

$$PR \triangleq \overline{hello}.\overline{coin}.coffee.\overline{drink}.PR$$

$$UNI \triangleq (CM \mid PR)\backslash\{coin, coffee\}$$

Specification

$$Spec \triangleq \overline{hello}.\tau.\tau.\overline{drink}.Spec$$

- We are given an abstract system specification *Spec*
- We devise an implementation *Imp* by assembling many interacting components

Are the processes *Imp* and *Spec* "behaviourally equivalent"?

- Fix a "good" notion of equivalence
- Prove that the two processes equivalent or find a counterexample and re-design *Imp*

Which Equivalence (1 / 2)?

What could be a reasonable equivalence relation?

1. Two processes are equivalent if their parse trees are identical
   - e.g., $P + Q + R = (P + Q) + R$!
   - …but this fails to capture the intuition that $P + Q = Q + P$
2. Two processes are equivalent up to renaming of the defining constants
   - e.g., $X \triangleq a.X$ is equivalent to $Y \triangleq a.Y$
3. Two processes are equivalent if the exhibit the same behaviour, i.e., if they give rise to the same LTS
   - …but this yields too many distinctions:

     $$X \triangleq a.X \qquad Y \triangleq a.a.Y \qquad Z \triangleq a.a.a.Z$$

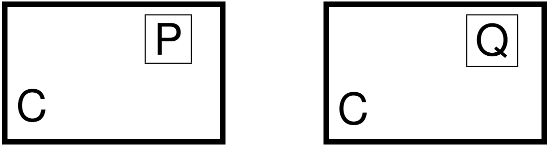
     have different LTSs but both processes can (only) execute infinitely many a-actions, and should be considered equivalent.

Which Equivalence (2 / 2)?

What should a reasonable behavioural equivalence satisfy?

- Abstracts from states (consider only the actions)
- Abstracts from internal behaviour (τ steps are not visible)
- Identifies processes whose LTSs are isomorphic
- Considers two processes equivalent only if both can execute the same actions sequences
- Allows to replace a subprocess by an equivalent counterpart without changing the overall semantics of the system
- Be deadlock sensitive, i.e., if one has a deadlock after a given trace s, then then the other process has a deadlock after the same trace (and vice versa).

Congruence

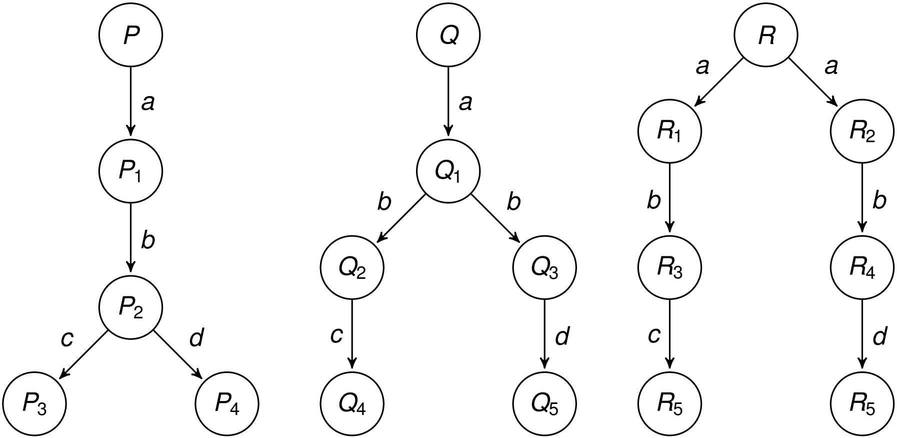


$C(P)$ $C(Q)$

Congruence Property

$P \equiv Q$ implies that $C(P) \equiv C(Q)$

Behavioural Equivalences



Problem: Are these three systems equivalent?

Trace Equivalence

Let $(Q, A, \rightarrow)$ be an LTS, with $q \in Q$.

Traces

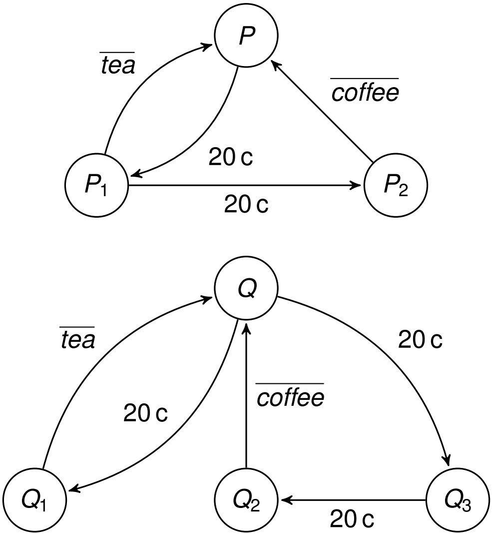
Let $s = a_1 \, a_2 \cdots a_k \in A^*$, for any $k \geq 1$, be a trace of q if there exists a sequence of transitions $q \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} q_k$,
with $q_i \in Q$ for all $1 \leq i \leq k$.

Let $T(q)$ be the set of all traces of state q.

Trace Equivalence

Two states p and q are *trace equivalent*, written $p =_T q$, if $T(p) = T(q)$.

Two Trace-Equivalent Systems



Trace Equivalence and Process Algebra

Consider two trace-equivalent versions of a vending machine:

$$VM_1 \triangleq coin.(\overline{coffee}.VM_1 + \overline{tea}.VM_1)\ ,$$
$$VM_2 \triangleq coin.\overline{coffee}.VM_2 + coin.\overline{tea}.VM_2\ .$$

Allow each machine to interact with a user who wishes to have only coffee:

$$User \triangleq \overline{coin}.coffee.User$$

Consider now the two systems

$$\left(User \mid VM_1\right) \setminus \{coin, coffee, tea\}\ ,$$
$$\left(User \mid VM_2\right) \setminus \{coin, coffee, tea\}\ .$$

Question

Are $\left(User \mid VM_1\right) \setminus \{coin, coffee, tea\}$ and $\left(User \mid VM_2\right) \setminus \{coin, coffee, tea\}$ trace equivalent?

Solution

$$VM_1 \triangleq coin.(\overline{coffee}.VM_1 + \overline{tea}.VM_1)$$
$$VM_2 \triangleq coin.\overline{coffee}.VM_2 + coin.\overline{tea}.VM_2$$
$$User \triangleq \overline{coin}.coffee.User$$

VM_1 serves coffee:

$$\begin{aligned}(User \mid VM_1) \setminus \{coin, coffee, tea\} &\xrightarrow{\tau} \\ (coffee.User \mid (\overline{coffee}.VM_1 + \overline{tea}.VM_1)) \setminus \{coin, coffee, tea\} &\xrightarrow{\tau} \\ (User \mid VM_1) \setminus \{coin, coffee, tea\} &.\end{aligned}$$

VM_2 may steal the coin:

$$\begin{aligned}(User \mid VM_2) \setminus \{coin, coffee, tea\} &\xrightarrow{\tau} \\ (coffee.User \mid (\overline{tea}.VM_2)) \setminus \{coin, coffee, tea\} &\not\rightarrow \quad .\end{aligned}$$